**1.Implementing the Singleton Pattern**

**Logger.java**

public class Logger {

private static instance

private static Logger instance;

private Logger() {

System.out.println("Logger initialized");

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("Log message: " + message);

}

}

**TestLogger.java**

public class TestLogger {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("This is the first message.");

logger2.log("This is the second message.");

if (logger1 == logger2) {

System.out.println("Singleton verified: Both logger instances are the same.");

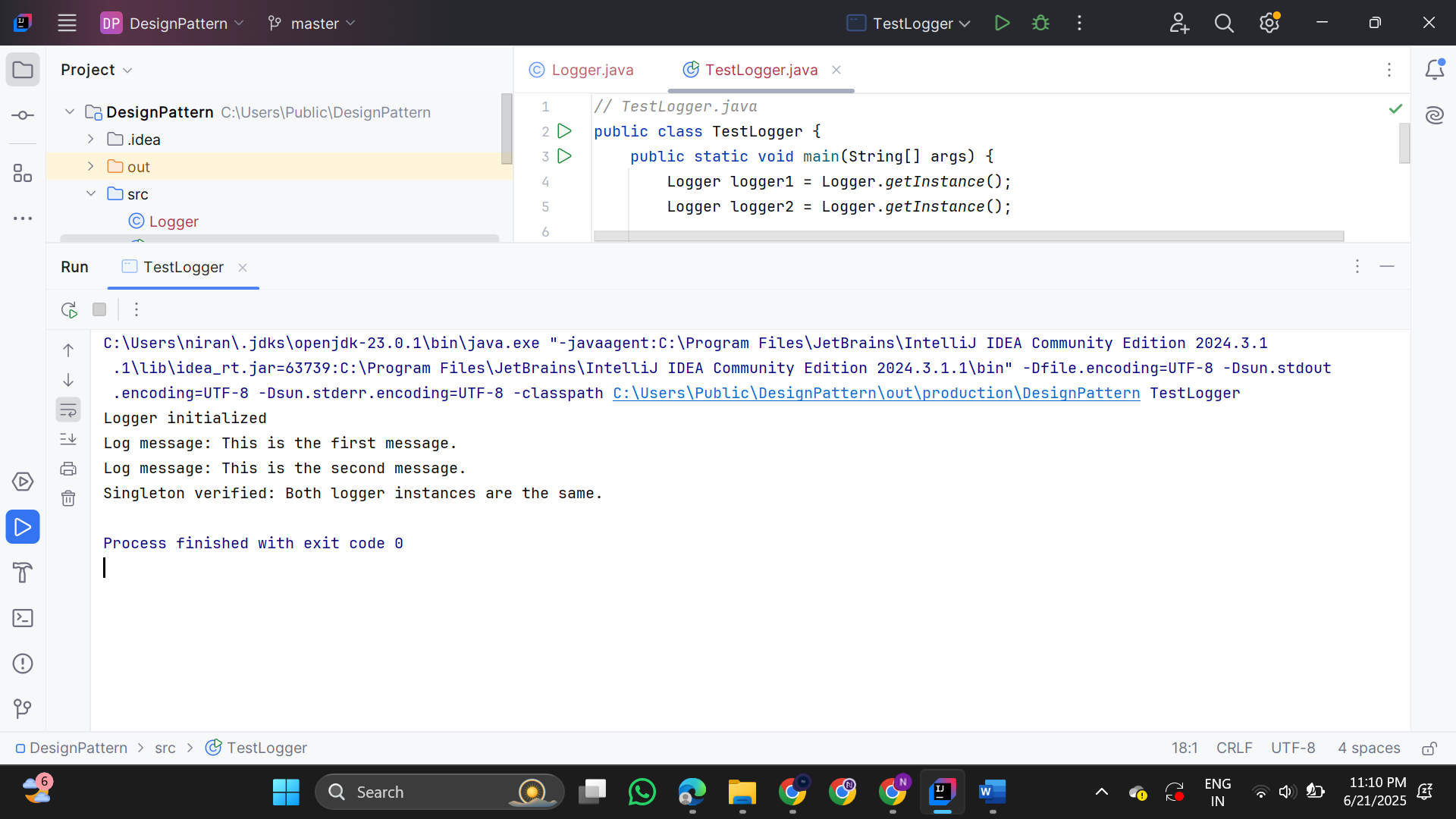
} else {

System.out.println("Singleton failed: Logger instances are different.");

}

}

}



**2. Implementing the Factory Method Pattern**

**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

public void open() {

System.out.println("Opening Word document...");

}

}

**PdfDocument.java**

public class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF document...");

}

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

public void open() {

System.out.println("Opening Excel document...");

}

}

**DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory

{

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**DocumentTest.java**

public class DocumentTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

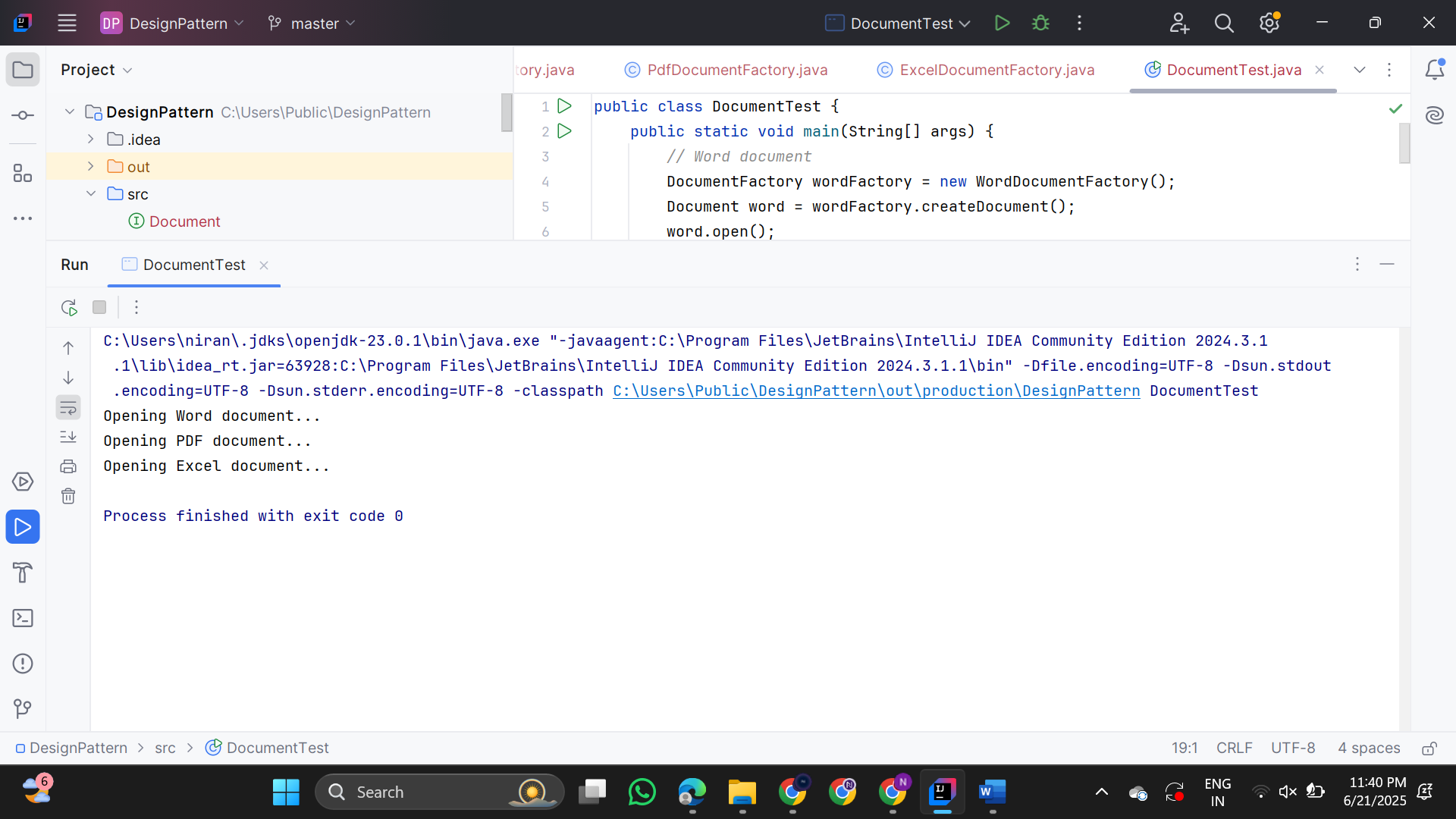
pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}   
  
  


**3.Implement the Builder Pattern  
  
Computer.java**

public class Computer

{

private String CPU;

private String RAM;

private String storage;

private boolean hasGraphicsCard;

private boolean hasWiFi;

private Computer(Builder builder)

{

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.hasGraphicsCard = builder.hasGraphicsCard;

this.hasWiFi = builder.hasWiFi;

}

public static class Builder

{

private String CPU;

private String RAM;

private String storage;

private boolean hasGraphicsCard;

private boolean hasWiFi;

public Builder setCPU(String CPU)

{

this.CPU = CPU;

return this;

}

public Builder setRAM(String RAM)

{

this.RAM = RAM;

return this;

}

public Builder setStorage(String storage)

{

this.storage = storage;

return this;

}

public Builder setGraphicsCard(boolean hasGraphicsCard) {

this.hasGraphicsCard = hasGraphicsCard;

return this;

}

public Builder setWiFi(boolean hasWiFi)

{

this.hasWiFi = hasWiFi;

return this;

}

public Computer build()

{

return new Computer(this);

}

}

@Override

public String toString()

{

return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage +

", GraphicsCard=" + hasGraphicsCard + ", WiFi=" + hasWiFi + "]";

}

}

**ComputerTest.java**

public class ComputerTest {

public static void main(String[] args) {

Computer gamingPC = new Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard(true)

.setWiFi(true)

.build();

Computer officePC = new Computer.Builder()

.setCPU("Intel i3")

.setRAM("8GB")

.setStorage("512GB HDD")

.setGraphicsCard(false)

.setWiFi(false)

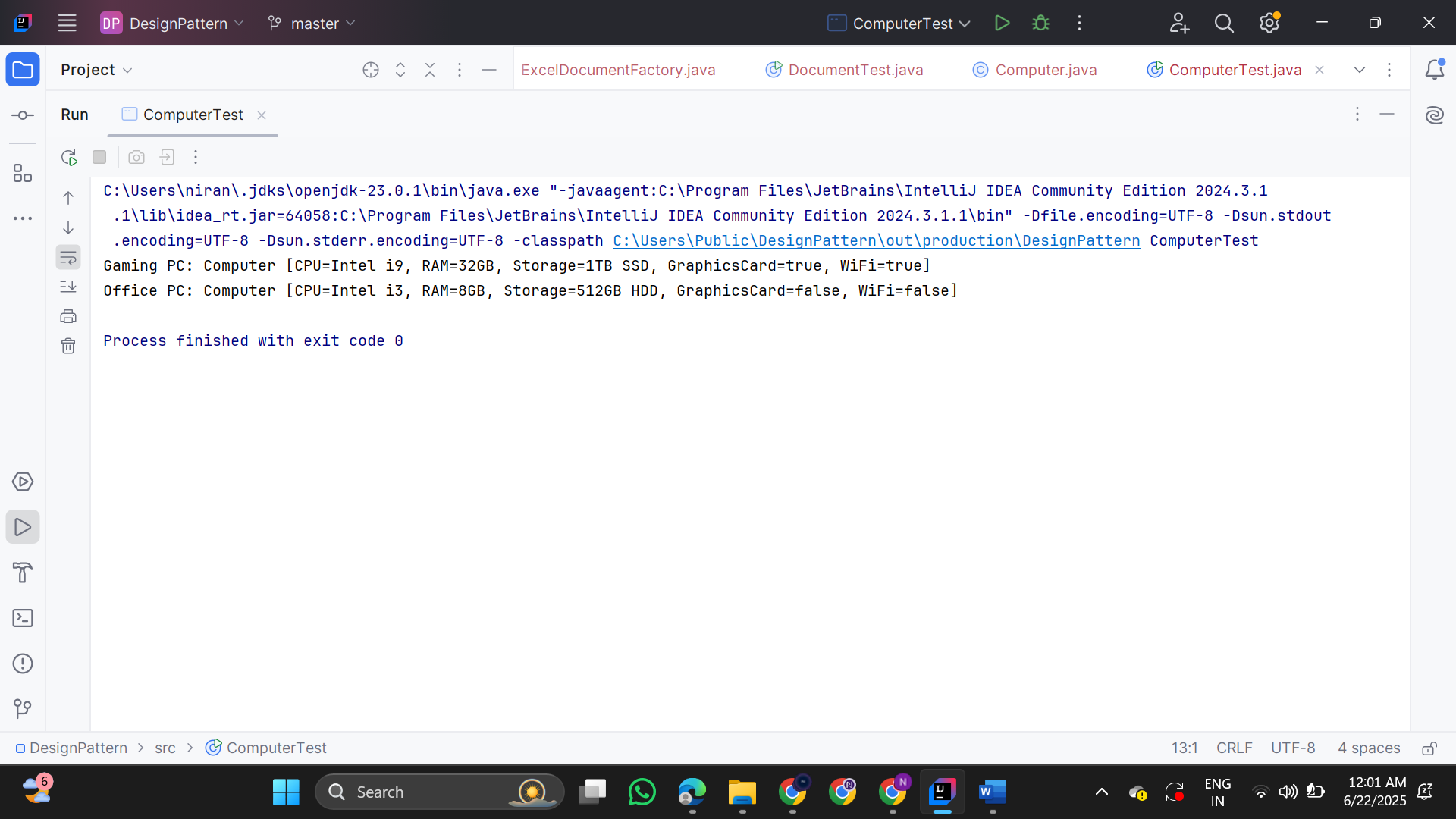
.build();

System.out.println("Gaming PC: " + gamingPC);

System.out.println("Office PC: " + officePC);

}

}



**4. Implementing the Adapter Pattern  
  
PaymentProcessor.java**

public interface PaymentProcessor {

void processPayment(double amount);

}

**PayPalGateway.java**

public class PayPalGateway {

public void sendPayment(double amount) {

System.out.println("Processing PayPal payment of $" + amount);

}

}

**StripeGateway.java**

public class StripeGateway {

public void makePayment(double value) {

System.out.println("Processing Stripe payment of $" + value);

}

}

**RazorpayGateway.java**

public class RazorpayGateway {

public void doTransaction(double money) {

System.out.println("Processing Razorpay transaction of $" + money);

}

}

**PayPalAdapter.java**

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway paypal;

public PayPalAdapter(PayPalGateway paypal) {

this.paypal = paypal;

}

@Override

public void processPayment(double amount) {

paypal.sendPayment(amount);

}

}

**StripeAdapter.java**

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripe;

public StripeAdapter(StripeGateway stripe) {

this.stripe = stripe;

}

@Override

public void processPayment(double amount) {

stripe.makePayment(amount);

}

}

**RazorpayAdapter.java**

public class RazorpayAdapter implements PaymentProcessor {

private RazorpayGateway razorpay;

public RazorpayAdapter(RazorpayGateway razorpay) {

this.razorpay = razorpay;

}

@Override

public void processPayment(double amount) {

razorpay.doTransaction(amount);

}

}

**PaymentTest.java**

public class PaymentTest {

public static void main(String[] args) {

PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());

PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());

PaymentProcessor razorpayProcessor = new RazorpayAdapter(new RazorpayGateway());

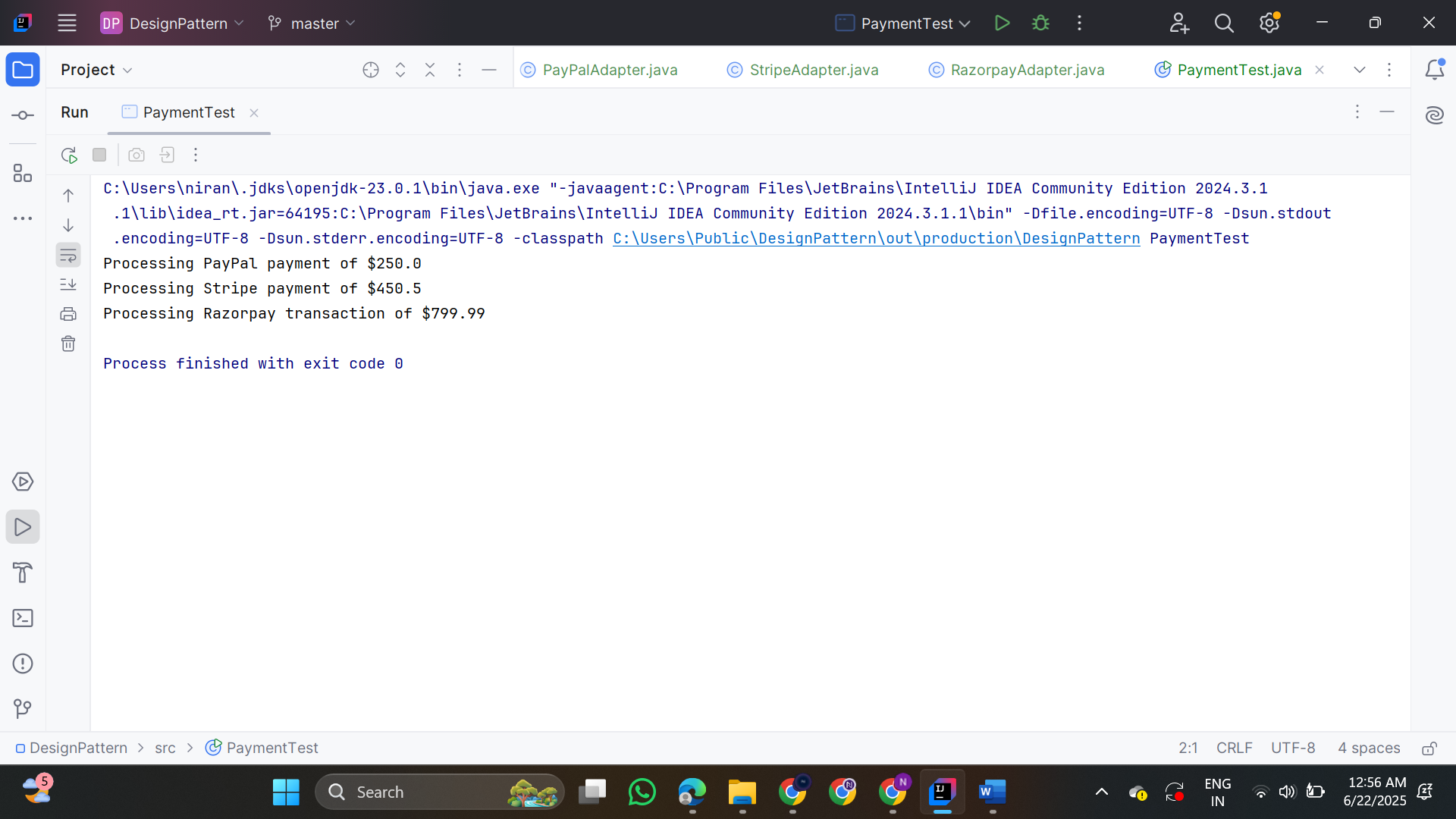
paypalProcessor.processPayment(250.0);

stripeProcessor.processPayment(450.5);

razorpayProcessor.processPayment(799.99);

}

}



**5. Implementing the Decorator Pattern  
  
Notifier.java**

public interface Notifier {

void send(String message);

}

**EmailNotifier.java**

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

System.out.println("Sending Email: " + message);

}

}

**NotifierDecorator.java**

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

@Override

public void send(String message) {

wrappedNotifier.send(message);

}

}

**SMSNotifierDecorator.java**

public class SMSNotifierDecorator extends NotifierDecorator   
{

public SMSNotifierDecorator(Notifier notifier)  
 {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("Sending SMS: " + message);

}

}

**SlackNotifierDecorator.java**

public class SlackNotifierDecorator extends NotifierDecorator

{   
public SlackNotifierDecorator(Notifier notifier)   
{

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("Sending Slack message: " + message);

}

}

**NotificationTest.java**

public class NotificationTest   
{

public static void main(String[] args)   
{

Notifier notifier = new EmailNotifier();

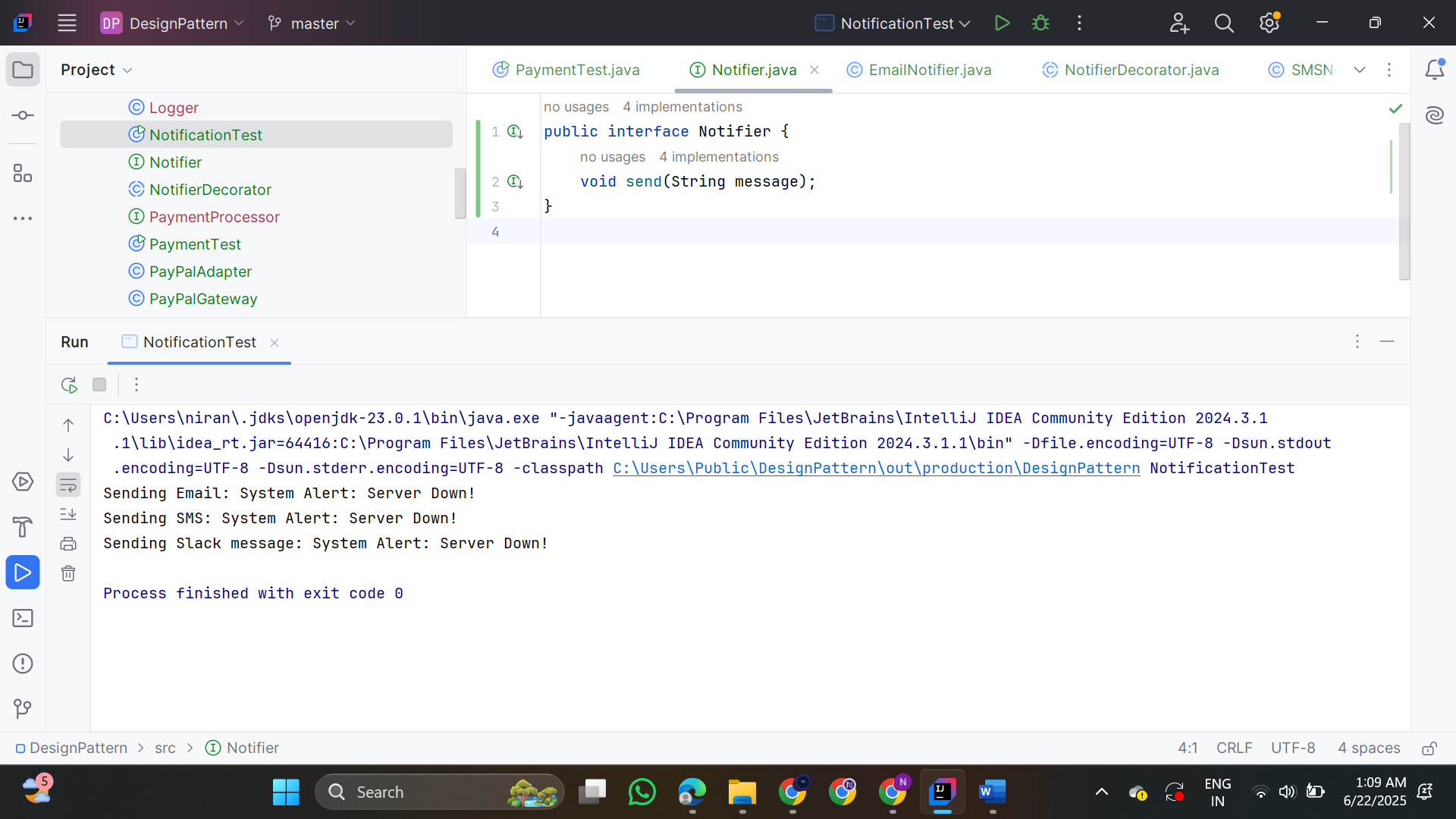
notifier = new SMSNotifierDecorator(notifier);

notifier = new SlackNotifierDecorator(notifier);

notifier.send("System Alert: Server Down!");

}

}



**6. Implementing the Proxy Pattern  
  
  
Image.java**

public interface Image   
{

void display();

}

**RealImage.java**

public class RealImage implements Image  
 {

private String fileName;

public RealImage(String fileName)  
 {

this.fileName = fileName;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

System.out.println("Loading " + fileName + " from remote server...");

}

@Override

public void display() {

System.out.println("Displaying " + fileName);

}

}

**ProxyImage.java**

public class ProxyImage implements Image {

private RealImage realImage;

private String fileName;

public ProxyImage(String fileName) {

this.fileName = fileName;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(fileName); }

realImage.display();

}

}

**ImageViewer.java**

public class ImageViewer {

public static void main(String[] args) {

Image image1 = new ProxyImage("photo1.jpg");

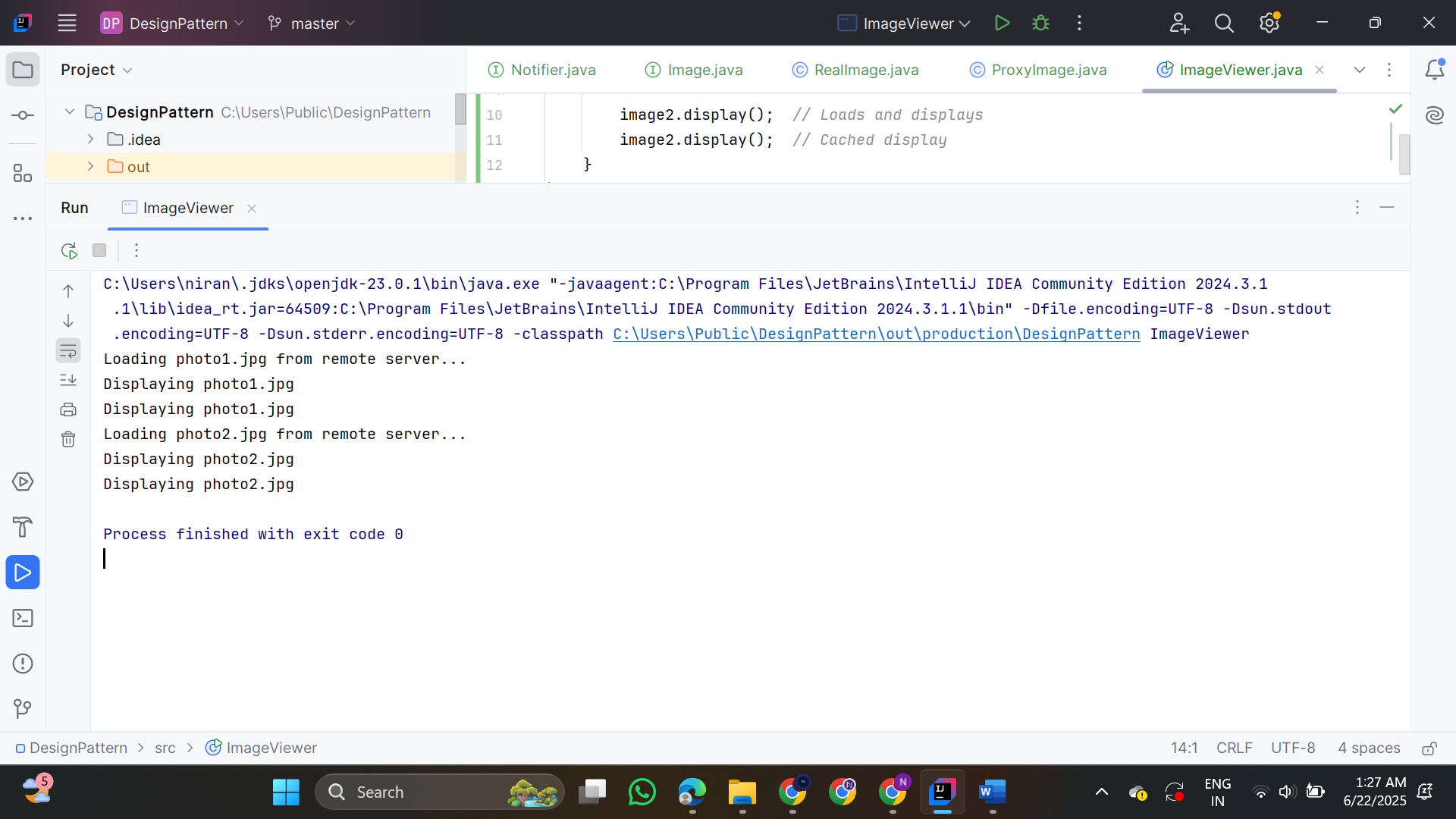
Image image2 = new ProxyImage("photo2.jpg");

image1.display();

image1.display();

image2.display();

image2.display(); } }



**7. Implementing the Observer Pattern**  
  
**Stock.java**

public interface Stock {

void registerObserver(Observer observer);

void removeObserver(Observer observer);

void notifyObservers();

}

**StockMarket.java**

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers = new ArrayList<>();

private double stockPrice;

@Override

public void registerObserver(Observer observer) {

observers.add(observer);

}

@Override

public void removeObserver(Observer observer) {

observers.remove(observer);

}

@Override

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockPrice);

}

}

public void setStockPrice(double price) {

this.stockPrice = price;

System.out.println("\nStock price updated to: $" + price);

notifyObservers();

}

}

**Observer.java**

public interface Observer {

void update(double stockPrice);

}

**MobileApp.java**

public class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

@Override

public void update(double stockPrice) {

System.out.println("[" + name + "] Mobile App received updated stock price: $" + stockPrice);

}

}

**WebApp.java**

public class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

@Override

public void update(double stockPrice) {

System.out.println("[" + name + "] Web App received updated stock price: $" + stockPrice);

}

}

**StockMonitorTest.java**

public class StockMonitorTest {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobileObserver = new MobileApp("ClientA");

Observer webObserver = new WebApp("ClientB");

stockMarket.registerObserver(mobileObserver);

stockMarket.registerObserver(webObserver);

stockMarket.setStockPrice(120.50);

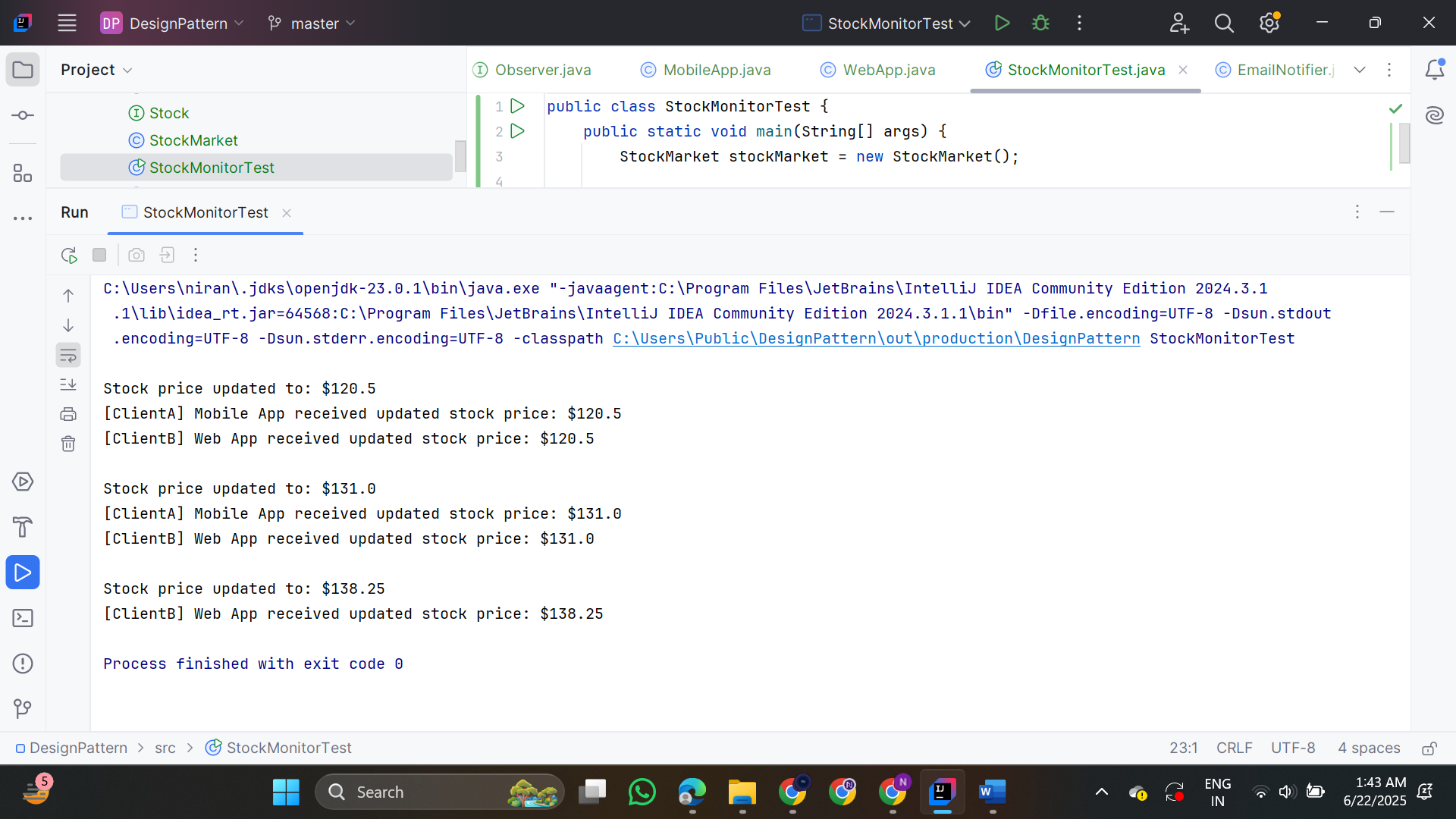
stockMarket.setStockPrice(131.00);

stockMarket.removeObserver(mobileObserver);

stockMarket.setStockPrice(138.25);

}

}



**8. Implementing the Strategy Pattern  
  
PaymentStrategy.java**

public interface PaymentStrategy {

void pay(double amount);

}

**CreditCardPayment.java**

public class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

public CreditCardPayment(String cardNumber) {

this.cardNumber = cardNumber;

}

@Override

public void pay(double amount) {

System.out.println("Paid $" + amount + " using Credit Card [" + cardNumber + "]");

}

}

**PayPalPayment.java**

public class PayPalPayment implements PaymentStrategy {

private String email;

public PayPalPayment(String email) {

this.email = email;

}

@Override

public void pay(double amount) {

System.out.println("Paid $" + amount + " using PayPal account [" + email + "]");

}

}

**PaymentContext.java**

public class PaymentContext {

private PaymentStrategy paymentStrategy;

// Allows changing payment method at runtime

public void setPaymentStrategy(PaymentStrategy paymentStrategy) {

this.paymentStrategy = paymentStrategy;

}

public void processPayment(double amount) {

if (paymentStrategy == null) {

System.out.println("No payment strategy selected.");

} else {

paymentStrategy.pay(amount);

}

}

}

**PaymentTest.java**

public class PaymentTest {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

// Use Credit Card

context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456"));

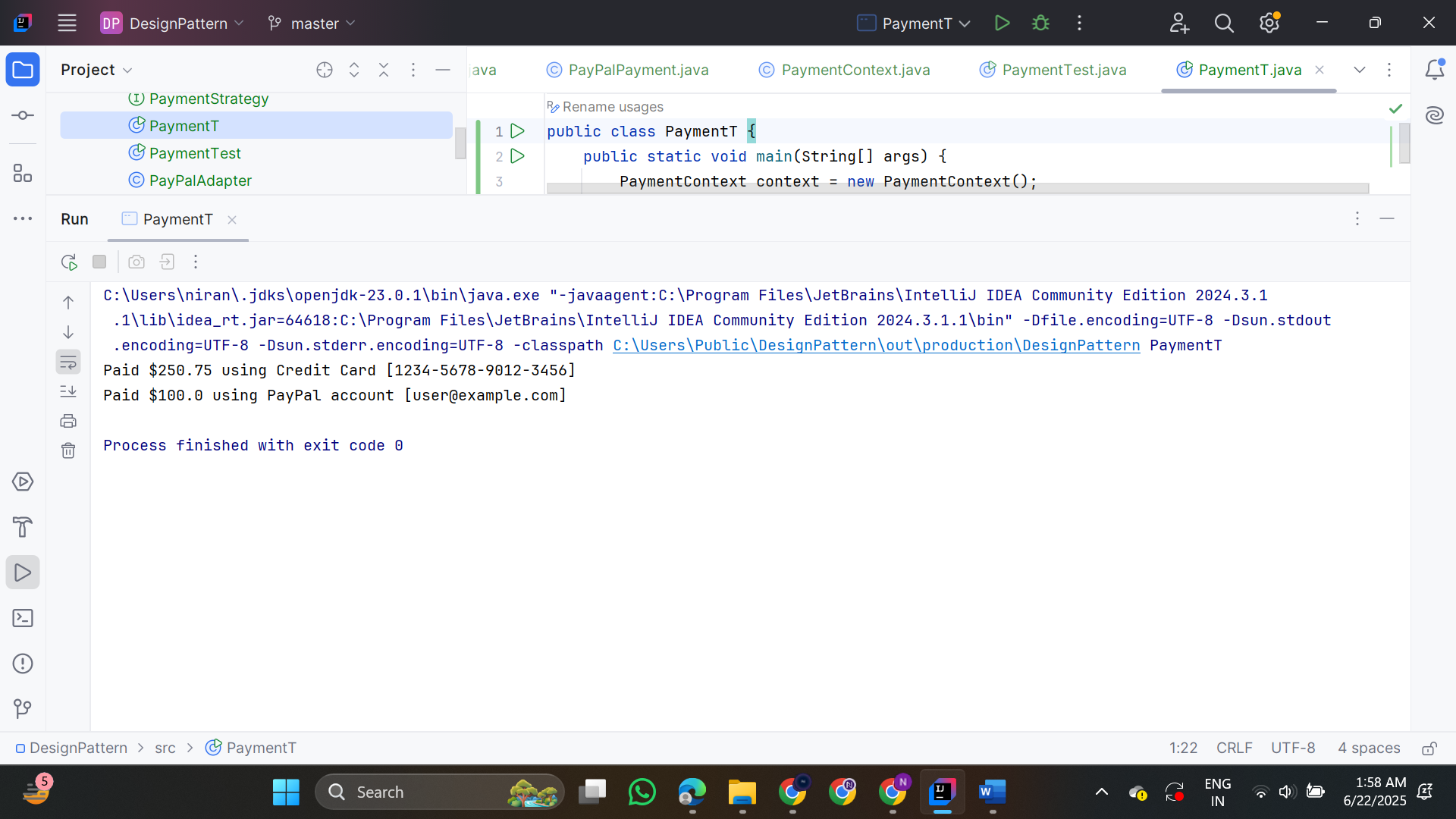
context.processPayment(250.75);

context.setPaymentStrategy(new PayPalPayment("user@example.com"));

context.processPayment(100.00);

}

}

 **9.Implementing the Command Pattern  
  
Command.java**

public interface Command {

void execute();

}

**LightOnCommand.java**

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOn();

}

}

**LightOffCommand.java**

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOff();

}

}

**RemoteControl.java**

public class RemoteControl {

private Command command;

// Set the command dynamically

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

if (command != null) {

command.execute();

} else {

System.out.println("No command assigned to button.");

}

}

}

**Light.java**

public class Light {

public void turnOn() {

System.out.println("Light is ON");

}

public void turnOff() {

System.out.println("Light is OFF");

}

}

**HomeAutomationTest.java**

public class HomeAutomationTest {

public static void main(String[] args) {

Light livingRoomLight = new Light();

Command lightOn = new LightOnCommand(livingRoomLight);

Command lightOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

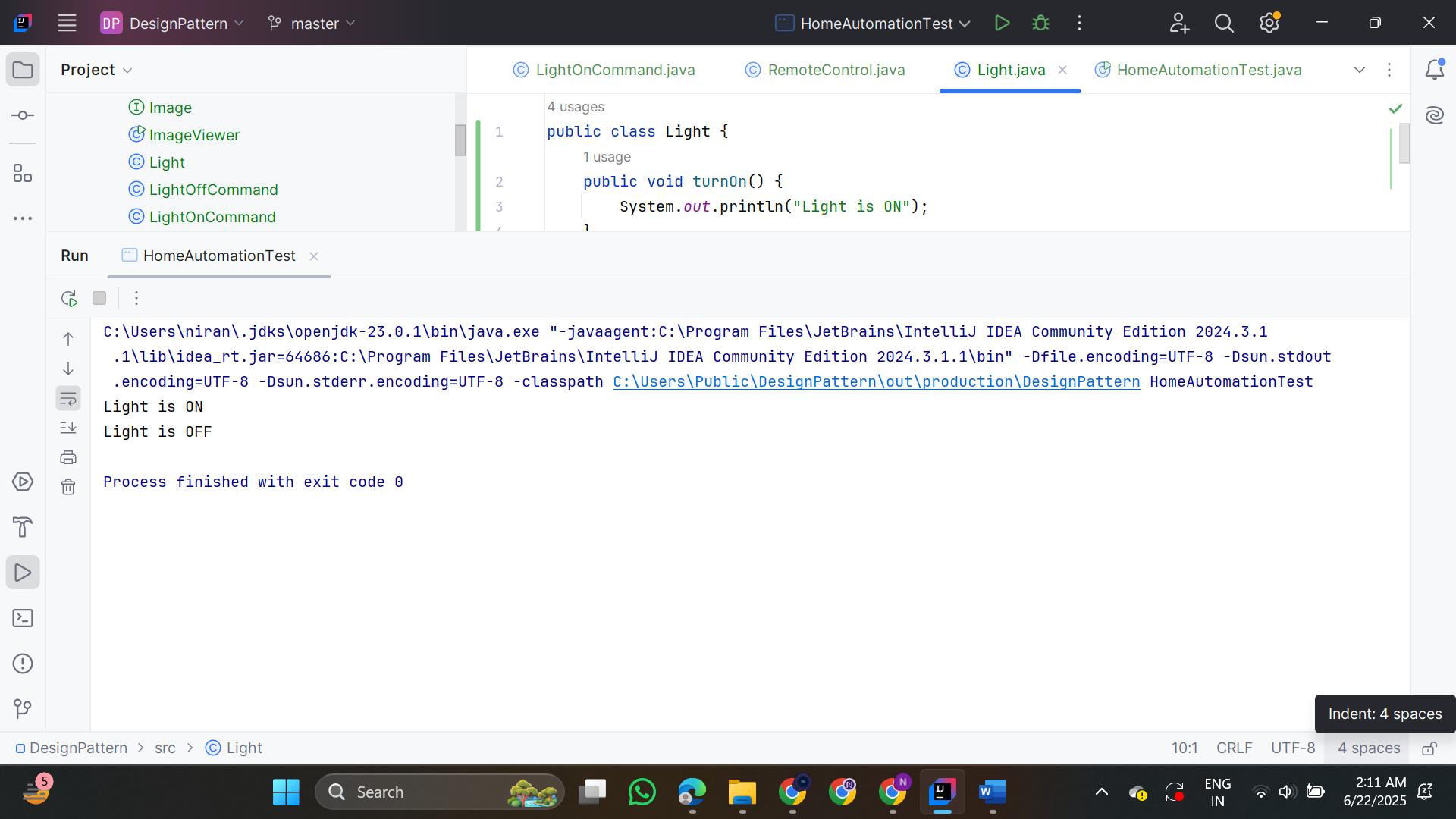
remote.setCommand(lightOn);

remote.pressButton();

remote.setCommand(lightOff);

remote.pressButton();

}

}   
  


**10. Implementing the MVC Pattern  
  
Student.java**

public class Student {

private String id;

private String name;

private String grade;

public Student(String id, String name, String grade) {

this.id = id;

this.name = name;

this.grade = grade;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

**StudentView.java**

public class StudentView {

public void displayStudentDetails(String id, String name, String grade) {

System.out.println("Student Details:");

System.out.println("ID : " + id);

System.out.println("Name : " + name);

System.out.println("Grade : " + grade);

}

}

**StudentController.java**

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void setStudentName(String name) {

model.setName(name);

}

public String getStudentName() {

return model.getName();

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public String getStudentGrade() {

return model.getGrade();

}

public void updateView() {

view.displayStudentDetails(model.getId(), model.getName(), model.getGrade());

}

}

**MVCTest.java**

public class MVCTest {

public static void main(String[] args) {

Student student = new Student("101", "Alice", "A");

StudentView view = new StudentView();   
  
StudentController controller = new StudentController(student, view);

controller.updateView();

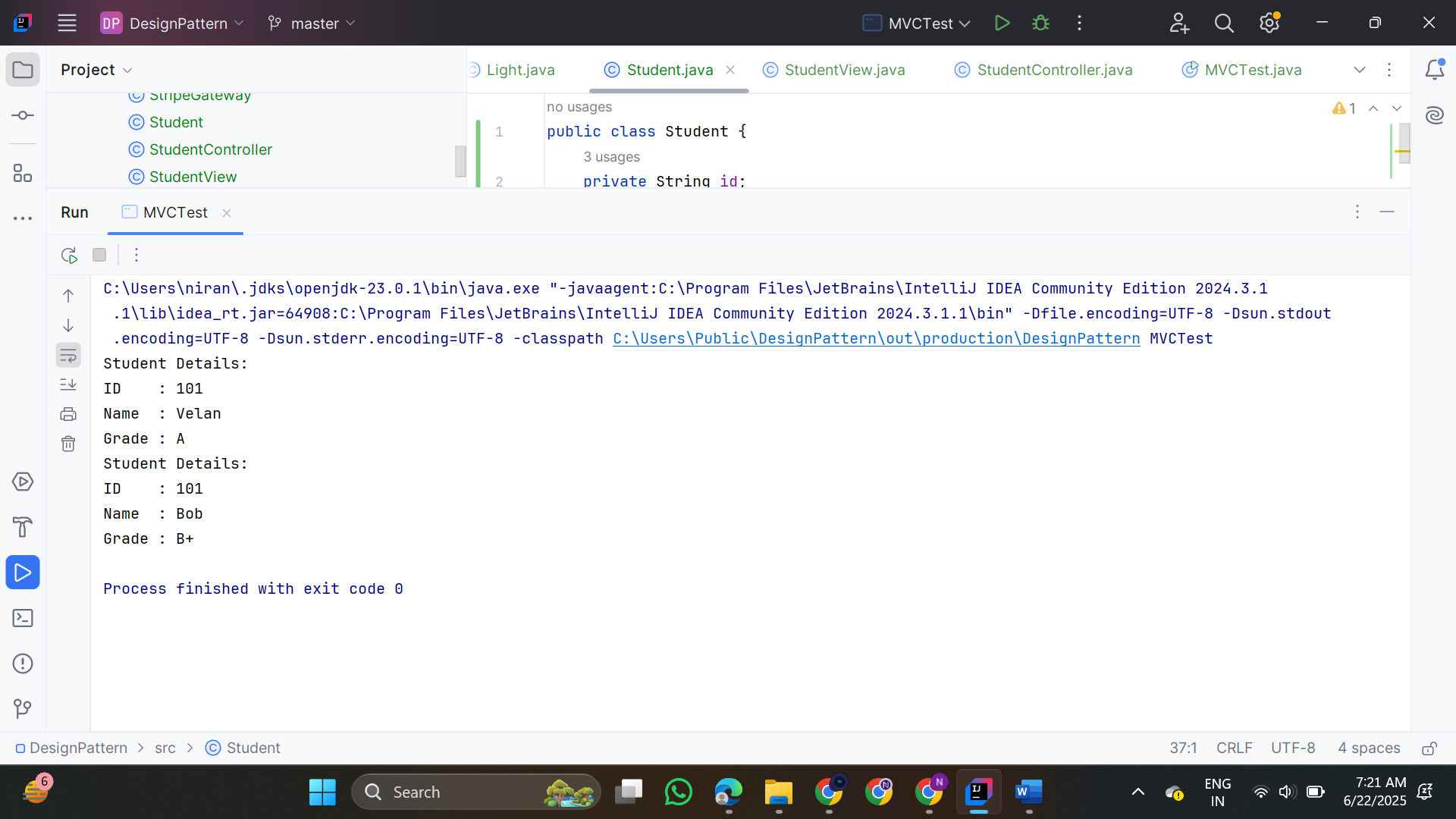
controller.setStudentName("Bob");

controller.setStudentGrade("B+");

controller.updateView();

}

}

  
  
  
 **11.Dependency Injection  
  
CustomerRepository.java**

public interface CustomerRepository {

String findCustomerById(String customerId);

}

**CustomerRepositoryImpl.java**

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public String findCustomerById(String customerId) {   
  
return "Customer[ID=" + customerId + ", Name=Alice]";

}

}

**CustomerService.java**

public class CustomerService {

private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public void getCustomerDetails(String customerId) {

String customer = customerRepository.findCustomerById(customerId);

System.out.println("Customer Details: " + customer);

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

service.getCustomerDetails("C001");

}

}

